

Showing invariance compositionally for a process algebra for network protocols

Timothy Bourke^{1,2}, Robert J. van Glabbeek^{3,4}, and Peter Höfner^{3,4}

¹ Inria Paris-Rocquencourt

² Ecole normale supérieure, Paris, France

³ NICTA, Sydney, Australia

⁴ Computer Science and Engineering, UNSW, Sydney, Australia

Abstract. This paper presents the mechanization of a process algebra for Mobile Ad hoc Networks and Wireless Mesh Networks, and the development of a compositional framework for proving invariant properties. Mechanizing the core process algebra in Isabelle/HOL is relatively standard, but its layered structure necessitates special treatment. The control states of reactive processes, such as nodes in a network, are modelled by terms of the process algebra. We propose a technique based on these terms to streamline proofs of inductive invariance. This is not sufficient, however, to state and prove invariants that relate states across multiple processes (entire networks). To this end, we propose a novel compositional technique for lifting global invariants stated at the level of individual nodes to networks of nodes.

1 Introduction and related work

The Algebra for Wireless Networks (AWN) is a process algebra developed for modelling and analysing protocols for Mobile Ad hoc Networks (MANETs) and Wireless Mesh Networks (WMNs) [6, §4]. This paper reports on both its mechanization in Isabelle/HOL [15] and the development of a compositional framework for showing invariant properties of models.¹ The techniques we describe are a response to problems encountered during the mechanization of a model and proof—presented elsewhere [4]—of an RFC-standard for routing protocols. Despite the existence of extensive research on related problems [18] and several mechanized frameworks for reactive systems [5, 10, 14], we are not aware of other solutions that allow the compositional statement and proof of properties relating the states of different nodes in a message-passing model—at least not within the strictures imposed by an Interactive Theorem Prover (ITP).

But is there really any need for yet another process algebra and associated framework? AWN provides a unique mix of communication primitives and a treatment of data structures that are essential for studying MANET and WMN protocols with dynamic topologies and sophisticated routing logic [6, §1]. It supports communication primitives for one-to-one (*unicast*), one-to-many (*groupcast*), and one-to-all (*broadcast*) message passing. AWN comprises distinct layers

¹ The Isabelle/HOL source files can be found in the Archive of Formal Proofs [3].

for expressing the structure of nodes and networks. We exploit this structure, but we also expect the techniques proposed in Sections 3 and 4 to apply to similar layered modelling languages. Besides this, our work differs from other mechanizations for verifying reactive systems, like UNITY [10], TLA^+ [5], or I/O Automata [14] (from which we drew the most inspiration), in its explicit treatment of control states, in the form of process algebra terms, as distinct from data states. In this respect, our approach is close to that of Isabelle/Circus [7], but it differs in (1) the treatment of operators for composing nodes, which we model directly as functions on automata, (2) the treatment of recursive invocations, which we do not permit, and (3) our inclusion of a framework for compositional proofs. Other work in ITPs focuses on showing traditional properties of process algebras, like, for instance, the treatment of binders [1], that bisimulation equivalence is a congruence [9, 11], or properties of fix-point induction [20], while we focus on what has been termed ‘proof methodology’ [8], and develop a compositional method for showing correctness properties of protocols specified in a process algebra. Alternatively, Paulson’s inductive approach [16] can be applied to show properties of protocols specified with less generic infrastructure. But we think it to be better suited to systems specified in a ‘declarative’ style as opposed to the strongly operational models we consider.

Structure and contributions. Section 2 describes the mechanization of AWN. The basic definitions are routine but the layered structure of the language and the treatment of operators on networks as functions on automata are relatively novel and essential to understanding later sections. Section 3 describes our mechanization of the theory of inductive invariants, closely following [13]. We exploit the structure of AWN to generate verification conditions corresponding to those of pen-and-paper proofs [6, §7]. Section 4 presents a compositional technique for stating and proving invariants that relate states across multiple nodes. Basically, we substitute ‘open’ Structural Operational Semantics (SOS) rules over the global state for the standard rules over local states (Section 4.1), show the property over a single sequential process (Section 4.2), ‘lift’ it successively over layers that model message queueing and network communication (Section 4.3), and, ultimately, ‘transfer’ it to the original model (Section 4.4).

2 The process algebra AWN

AWN comprises five layers [6, §4]. We treat each layer as an automaton with states of a specific form and a given set of transition rules. We describe the layers from the bottom up over the following sections.

2.1 Sequential processes

Sequential processes are used to encode protocol logic. Each is modelled by a (recursive) specification Γ of type $\text{'p} \Rightarrow (\text{'s}, \text{'p}, \text{'l}) \text{ seqp}$, which maps process names of type 'p to terms of type $(\text{'s}, \text{'p}, \text{'l}) \text{ seqp}$, also parameterized by 's , data states, and 'l , labels. States of sequential processes have the form (ξ, p) where ξ is a data state of type 's and p is a control term of type $(\text{'s}, \text{'p}, \text{'l}) \text{ seqp}$.

$\{\}\llbracket u \rrbracket p$	$'l \Rightarrow ('s \Rightarrow 's) \Rightarrow ('s, 'p, 'l) \text{ seqp} \Rightarrow ('s, 'p, 'l) \text{ seqp}$
$\{\}\langle g \rangle p$	$'l \Rightarrow ('s \Rightarrow 's \text{ set}) \Rightarrow ('s, 'p, 'l) \text{ seqp} \Rightarrow ('s, 'p, 'l) \text{ seqp}$
$\{\}\text{unicast}(s_{ip}, s_{msg}) . p \triangleright q$	$'l \Rightarrow ('s \Rightarrow ip) \Rightarrow ('s \Rightarrow msg) \Rightarrow ('s, 'p, 'l) \text{ seqp} \Rightarrow ('s, 'p, 'l) \text{ seqp}$
$\{\}\text{broadcast}(s_{msg}) . p$	$'l \Rightarrow ('s \Rightarrow msg) \Rightarrow ('s, 'p, 'l) \text{ seqp} \Rightarrow ('s, 'p, 'l) \text{ seqp}$
$\{\}\text{groupcast}(s_{ips}, s_{msg}) . p$	$'l \Rightarrow ('s \Rightarrow ip \text{ set}) \Rightarrow ('s \Rightarrow msg) \Rightarrow ('s, 'p, 'l) \text{ seqp} \Rightarrow ('s, 'p, 'l) \text{ seqp}$
$\{\}\text{send}(s_{msg}) . p$	$'l \Rightarrow ('s \Rightarrow msg) \Rightarrow ('s, 'p, 'l) \text{ seqp} \Rightarrow ('s, 'p, 'l) \text{ seqp}$
$\{\}\text{receive}(u_{msg}) . p$	$'l \Rightarrow (msg \Rightarrow 's \Rightarrow 's) \Rightarrow ('s, 'p, 'l) \text{ seqp} \Rightarrow ('s, 'p, 'l) \text{ seqp}$
$\{\}\text{deliver}(s_{data}) . p$	$'l \Rightarrow ('s \Rightarrow data) \Rightarrow ('s, 'p, 'l) \text{ seqp} \Rightarrow ('s, 'p, 'l) \text{ seqp}$
$p_1 \oplus p_2$	$('s, 'p, 'l) \text{ seqp} \Rightarrow ('s, 'p, 'l) \text{ seqp} \Rightarrow ('s, 'p, 'l) \text{ seqp}$
$\text{call}(pn)$	$'p \Rightarrow ('s, 'p, 'l) \text{ seqp}$

(a) Term constructors for $('s, 'p, 'l) \text{ seqp}$.

$\frac{\xi' = u \xi}{((\xi, \{\}\llbracket u \rrbracket p), \tau, (\xi', p)) \in \text{seqp-sos } \Gamma}$	$\frac{((\xi, p), a, (\xi', p')) \in \text{seqp-sos } \Gamma}{((\xi, p \oplus q), a, (\xi', p')) \in \text{seqp-sos } \Gamma}$
$\frac{((\xi, \Gamma pn), a, (\xi', p')) \in \text{seqp-sos } \Gamma}{((\xi, \text{call}(pn)), a, (\xi', p')) \in \text{seqp-sos } \Gamma}$	$\frac{((\xi, q), a, (\xi', q')) \in \text{seqp-sos } \Gamma}{((\xi, p \oplus q), a, (\xi', q')) \in \text{seqp-sos } \Gamma}$
$((\xi, \{\}\text{unicast}(s_{ip}, s_{msg}) . p \triangleright q), \text{unicast } (s_{ip} \xi) (s_{msg} \xi), (\xi, p)) \in \text{seqp-sos } \Gamma$	
$((\xi, \{\}\text{unicast}(s_{ip}, s_{msg}) . p \triangleright q), \neg \text{unicast } (s_{ip} \xi), (\xi, q)) \in \text{seqp-sos } \Gamma$	

(b) SOS rules for sequential processes: examples from seqp-sos .

Fig. 1: Sequential processes: terms and semantics

Process terms are built from the constructors that are shown with their types² in Figure 1a. The inductive set seqp-sos , shown partially in Figure 1b, contains one or two SOS rules for each constructor. It is parameterized by a specification Γ and relates triples of source states, actions, and destination states.

The ‘prefix’ constructors are each labelled with an $\{\}$. Labels are used to strengthen invariants when a property is only true in or between certain states; they have no influence on control flow (unlike in [13]). The prefix constructors are *assignment*, $\{\}\llbracket u \rrbracket p$, which transforms the data state deterministically according to the function u and performs a τ action, as shown in Figure 1b; *guard/bind*, $\{\}\langle g \rangle p$, with which we encode both guards, $\langle \lambda \xi. \text{if } g \ \xi \text{ then } \{\xi\} \text{ else } \emptyset \rangle p$, and variable bindings, as in $\langle \lambda \xi. \{\xi(\text{no} := n) \mid n < 5\} \rangle p$;³ *network synchronizations*, *receive/unicast/broadcast/groupcast*, of which the rules for unicast are characteristic and shown in Figure 1b—the environment decides between a successful $\text{unicast } i \ m$ and an unsuccessful $\neg \text{unicast } i$; and, *internal communications*, *send/receive/deliver*.

The other constructors are unlabelled and serve to ‘glue’ processes together: *choice*, $p_1 \oplus p_2$, takes the union of two transition sets; and, *call*, $\text{call}(pn)$, affixes a term from the specification (Γpn) . The rules for both are shown in Figure 1b.

² Leading abstractions are omitted, for example, $\lambda l \text{ fa } p. \{\}\llbracket u \rrbracket p$ is written $\{\}\llbracket u \rrbracket p$.

³ Although it strictly subsumes assignment we prefer to keep both.

We introduce the specification of a simple ‘toy’ protocol as a running example:

$$\begin{array}{ll}
\Gamma_{\text{Toy}} \text{PToy} = \text{labelled PTToy} (& \text{receive}(\lambda \text{msg}' \xi. \xi \mid \text{msg} := \text{msg}' \mid)) & \{\text{PToy-:0}\} \\
& \llbracket \lambda \xi. \xi \mid \text{nhip} := \text{ip } \xi \rrbracket & \{\text{PToy-:1}\} \\
& (& \langle \text{is-newpkt} \rangle & \{\text{PToy-:2}\} \\
& \quad \llbracket \lambda \xi. \xi \mid \text{no} := \max(\text{no } \xi) (\text{num } \xi) \rrbracket & \{\text{PToy-:3}\} \\
& \quad \text{broadcast}(\lambda \xi. \text{pkt}(\text{no } \xi, \text{ip } \xi)). \text{Toy}() & \{\text{PToy-:4,5}\} \\
& \oplus \langle \text{is-pkt} \rangle & \{\text{PToy-:2}\} \\
& \quad (& \langle \lambda \xi. \text{if num } \xi \geq \text{no } \xi \text{ then } \{\xi\} \text{ else } \{\} \rangle & \{\text{PToy-:6}\} \\
& \quad \quad \llbracket \lambda \xi. \xi \mid \text{no} := \text{num } \xi \rrbracket & \{\text{PToy-:7}\} \\
& \quad \quad \llbracket \lambda \xi. \xi \mid \text{nhip} := \text{sip } \xi \rrbracket & \{\text{PToy-:8}\} \\
& \quad \quad \text{broadcast}(\lambda \xi. \text{pkt}(\text{no } \xi, \text{ip } \xi)). \text{Toy}() & \{\text{PToy-:9,10}\} \\
& \oplus \langle \lambda \xi. \text{if num } \xi < \text{no } \xi \text{ then } \{\xi\} \text{ else } \{\} \rangle & \{\text{PToy-:6}\} \\
& \quad \text{Toy}())) , & \{\text{PToy-:11}\}
\end{array}$$

where PTToy is the process name, is-newpkt and is-pkt are guards that unpack the contents of msg, and Toy() is an abbreviation that clears some variables before a call(PTToy). The function labelled associates its argument PTToy paired with a number to every prefix constructor. There are two types of messages: newpkt (data, dst), from which is-newpkt copies data to the variable num, and pkt (data, src), from which is-pkt copies data into num and src into sip.

The corresponding sequential model is an automaton—a record⁴ of two fields: a set of initial states and a set of transitions—parameterized by an address i:

$$\text{ptoy } i = \langle \text{init} = \{(\text{toy-init } i, \Gamma_{\text{Toy}} \text{PTToy})\}, \text{trans} = \text{seqp-sos } \Gamma_{\text{Toy}} \rangle,$$

where toy-init i yields the initial data state (ip = i, no = 0, nhip = i, msg = SOME x. True, num = SOME x. True, sip = SOME x. True). The last three variables are initialized to arbitrary values, as they are considered local—they are explicitly reinitialized before each call(PTToy). This is the biggest departure from the original definition of AWN; it simplifies the treatment of call, as we show in Section 3.1, and facilitates working with automata where variable locality makes little sense.

2.2 Local parallel composition

Message sending protocols must nearly always be input-enabled, that is, nodes should always be in a state where they can receive messages. To achieve this, and to model asynchronous message transmission, the protocol process is combined with a queue model, qmsg, that continually appends received messages onto an

$$\begin{array}{c}
\frac{(s, a, s') \in S \quad \bigwedge m. a \neq \text{receive } m}{((s, t), a, (s', t)) \in \text{parp-sos } S \ T} \quad \frac{(t, a, t') \in T \quad \bigwedge m. a \neq \text{send } m}{((s, t), a, (s, t')) \in \text{parp-sos } S \ T} \\
\\
\frac{(s, \text{receive } m, s') \in S \quad (t, \text{send } m, t') \in T}{((s, t), \tau, (s', t')) \in \text{parp-sos } S \ T}
\end{array}$$

Fig. 2: SOS rules for parallel processes: parp-sos.

⁴ The generic record has type ('s, 'a) automaton, where the type 's is the domain of states, here pairs of data records and control terms, and 'a is the domain of actions.

$$\begin{array}{c}
\frac{(s, \text{groupcast } D \ m, s') \in S}{(s_R^i, (R \cap D):*\text{cast}(m), s_R^i) \in \text{node-sos } S} \quad \frac{(s, \text{receive } m, s') \in S}{(s_R^i, \{i\} \neg \emptyset:\text{arrive}(m), s_R^i) \in \text{node-sos } S} \\
(s_R^i, \emptyset \neg \{i\}:\text{arrive}(m), s_R^i) \in \text{node-sos } S \quad (s_R^i, \text{connect}(i, i'), s_{R \cup \{i'\}}^i) \in \text{node-sos } S
\end{array}$$

Fig. 3: SOS rules for nodes: examples from `node-sos`.

internal list and offers to send the head message to the protocol process:
`ptoy i << qmsg`. The *local parallel* operator is a function over automata:

$$s \langle\langle t = (\text{init} = \text{init } s \times \text{init } t, \text{trans} = \text{parp-sos } (\text{trans } s) (\text{trans } t)) \rangle\rangle.$$

The rules for `parp-sos` are shown in Figure 2.

2.3 Nodes

At the node level, a local process `np` is wrapped in a layer that records its address `i` and tracks the set of neighbouring node addresses, initially R_i :

$$\langle i : np : R_i \rangle = (\text{init} = \{s_{R_i}^i \mid s \in \text{init } np\}, \text{trans} = \text{node-sos } (\text{trans } np)).$$

Node states are denoted s_R^i . Figure 3 presents rules typical of `node-sos`. Output network synchronizations, like `groupcast`, are filtered by the list of neighbours to become `*cast` actions. The $H \neg K:\text{arrive}(m)$ action—in Figure 3 instantiated as $\emptyset \neg \{i\}:\text{arrive}(m)$, and $\{i\} \neg \emptyset:\text{arrive}(m)$ —is used to model a message `m` received by nodes in `H` and not by those in `K`. The `connect(i, i')` adds node `i'` to the set of neighbours of node `i`; `disconnect(i, i')` works similarly.

2.4 Partial networks

Partial networks are specified as values of type `net-tree`, that is, as a node $\langle i; R_i \rangle$ with address `i` and a set of initial neighbours R_i , or a composition of two `net-trees` $p_1 \parallel p_2$. The function `pnet` maps such a value, together with the process `np i` to execute at each node `i`, here parameterized by an address, to an automaton:

$$\begin{aligned}
\text{pnet } np \langle i; R_i \rangle &= \langle i : np \ i : R_i \rangle \\
\text{pnet } np (p_1 \parallel p_2) &= (\text{init} = \{s_1 \parallel s_2 \mid s_1 \in \text{init } (\text{pnet } np \ p_1) \wedge s_2 \in \text{init } (\text{pnet } np \ p_2)\}, \\
&\quad \text{trans} = \text{pnet-sos } (\text{trans } (\text{pnet } np \ p_1)) (\text{trans } (\text{pnet } np \ p_2))) ,
\end{aligned}$$

The states of such automata mirror the tree structure of the network term; we denote composed states $s_1 \parallel s_2$. This structure, and the node addresses, remain constant during an execution. These definitions suffice to model an example three node network of toy processes:

$$\text{pnet } (\lambda i. \text{ptoy } i \langle\langle \text{qmsg} \rangle\rangle (\langle A; \{B\} \rangle \parallel \langle B; \{A, C\} \rangle \parallel \langle C; \{B\} \rangle)).$$

Figure 4 presents rules typical of `pnet-sos`. There are rules where only one node acts, like the one shown for τ , and rules where all nodes act, like those for `*cast` and `arrive`. The latter ensure—since `qmsg` is always ready to receive `m`—that a partial network can always perform an $H \neg K:\text{arrive}(m)$ for any combination of `H` and `K` consistent with its node addresses, but that pairing with an $R:\text{*cast}(m)$ restricts the possibilities to the one consistent with the destinations in `R`.

$$\begin{array}{c}
\frac{(s, R:\text{*cast}(m), s') \in S \quad (t, H \neg K:\text{arrive}(m), t') \in T \quad H \subseteq R \quad K \cap R = \emptyset}{(s \parallel t, R:\text{*cast}(m), s' \parallel t') \in \text{pnet-sos } S \ T} \\
\\
\frac{(s, H \neg K:\text{arrive}(m), s') \in S \quad (t, H' \neg K':\text{arrive}(m), t') \in T}{(s \parallel t, (H \cup H') \neg (K \cup K'):\text{arrive}(m), s' \parallel t') \in \text{pnet-sos } S \ T} \quad \frac{(s, \tau, s') \in S}{(s \parallel t, \tau, s' \parallel t) \in \text{pnet-sos } S \ T}
\end{array}$$

Fig. 4: SOS rules for partial networks: examples from pnet-sos.

2.5 Complete networks

The last layer closes a network to further interactions with an environment; the *cast action becomes a τ and $H \neg K:\text{arrive}(m)$ is forbidden:

$$\text{closed } A = A(\text{trans} := \text{cnet-sos } (\text{trans } A)).$$

The rules for cnet-sos are straight-forward and not presented here.

3 Basic invariance

This paper only considers proofs of invariance, that is, properties of reachable states. The basic definitions are classic [14, Part III].

Definition 1 (reachability). *Given an automaton A and an assumption I over actions, reachable $A \ I$ is the smallest set defined by the rules:*

$$\frac{s \in \text{init } A}{s \in \text{reachable } A \ I} \quad \frac{s \in \text{reachable } A \ I \quad (s, a, s') \in \text{trans } A \quad I \ a}{s' \in \text{reachable } A \ I}$$

Definition 2 (invariance). *Given an automaton A and an assumption I , a predicate P is invariant, denoted $A \models (I \rightarrow) P$, iff $\forall s \in \text{reachable } A \ I. P \ s$.*

We state reachability relative to an assumption on (input) actions I . When I is λ -. True, we write simply $A \models P$.

Definition 3 (step invariance). *Given an automaton A and an assumption I , a predicate P is step invariant, denoted $A \models (I \rightarrow) P$, iff*

$$\forall a. I \ a \longrightarrow (\forall s \in \text{reachable } A \ I. \forall s'. (s, a, s') \in \text{trans } A \longrightarrow P \ (s, a, s')).$$

Our invariance proofs follow the compositional strategy recommended in [18, §1.6.2]. That is, we show properties of sequential process automata using the induction principle of Definition 1, and then apply generic proof rules to successively lift such properties over each of the other layers. The inductive assertion method, as stated in rule INV-B of [13], requires a finite set of transition schemas, which, together with the obligation on initial states yields a set of sufficient verification conditions. We develop this set in Section 3.1 and use it to derive the main proof rule presented in Section 3.2 together with some examples.

3.1 Control terms

Given a specification Γ over finitely many process names, we can generate a finite set of verification conditions because transitions from ('s, 'p, 'l) **seqp** terms always yield subterms of terms in Γ . But, rather than simply consider the set of all subterms, we prefer to define a subset of ‘control terms’ that reduces the number of verification conditions, avoids tedious duplication in proofs, and corresponds with the obligations considered in pen-and-paper proofs. The main idea is that the \oplus and **call** operators serve only to combine process terms: they are, in a sense, executed recursively by **seqp-sos** to determine the actions that a term offers to its environment. This is made precise by defining a relation between sequential process terms.

Definition 4 (\sim_{Γ}). *For a (recursive) specification Γ , let \sim_{Γ} be the smallest relation such that $(p_1 \oplus p_2) \sim_{\Gamma} p_1$, $(p_1 \oplus p_2) \sim_{\Gamma} p_2$, and $(\text{call}(pn)) \sim_{\Gamma} \Gamma pn$.*

We write \sim_{Γ}^* for its reflexive transitive closure. We consider a specification to be *well formed*, when the inverse of this relation is well founded:

$$\text{wellformed } \Gamma = \text{wf } \{(q, p) \mid p \sim_{\Gamma} q\}.$$

Most of our lemmas only apply to well formed specifications, since otherwise functions over the terms they contain cannot be guaranteed to terminate. Neither of these two specifications is well formed: $\Gamma_a(1) = p \oplus \text{call}(1)$; $\Gamma_b(n) = \text{call}(n + 1)$.

We will also need a set of ‘start terms’—the subterms that can act directly.

Definition 5 (sterms). *Given a wellformed Γ and a sequential process term p , $\text{sterms } \Gamma p$ is the set of maximal elements related to p by the reflexive transitive closure of the \sim_{Γ} relation⁵:*

$$\begin{aligned} \text{sterms } \Gamma (p_1 \oplus p_2) &= \text{sterms } \Gamma p_1 \cup \text{sterms } \Gamma p_2, \\ \text{sterms } \Gamma (\text{call}(pn)) &= \text{sterms } \Gamma (\Gamma pn), \text{ and,} \\ \text{sterms } \Gamma p &= \{p\} \text{ otherwise.} \end{aligned}$$

We also define ‘local start terms’ by $\text{stermsl } (p_1 \oplus p_2) = \text{stermsl } p_1 \cup \text{stermsl } p_2$ and otherwise $\text{stermsl } p = \{p\}$ to permit the sufficient syntactic condition that a specification Γ is well formed if $\text{call}(pn) \notin \text{stermsl } (\Gamma pn)$.

Similarly to the way that start terms act as direct sources of transitions, we define ‘derivative terms’ giving possible active destinations of transitions.

Definition 6 (dterms). *Given a wellformed Γ and a sequential process term p , $\text{dterms } p$ is defined by:*

$$\begin{aligned} \text{dterms } \Gamma (p_1 \oplus p_2) &= \text{dterms } \Gamma p_1 \cup \text{dterms } \Gamma p_2, \\ \text{dterms } \Gamma (\text{call}(pn)) &= \text{dterms } \Gamma (\Gamma pn), \\ \text{dterms } \Gamma (\{l\}[u] p) &= \text{sterms } \Gamma p, \\ \text{dterms } \Gamma (\{l\}\text{unicast}(s_{ip}, s_{msg}) . p \triangleright q) &= \text{sterms } \Gamma p \cup \text{sterms } \Gamma q, \text{ and so on.} \end{aligned}$$

⁵ This characterization is equivalent to $\{q \mid p \sim_{\Gamma}^* q \wedge (\nexists q'. q \sim_{\Gamma} q')\}$. Termination follows from wellformed Γ , that is, $\text{wellformed } \Gamma \implies \text{sterms-dom } (\Gamma, p)$ for all p .

These derivative terms overapproximate the set of reachable terms, since they do not consider the truth of guards nor the willingness of communication partners.

These auxiliary definitions lead to a succinct definition of the set of control terms of a specification.

Definition 7 (cterm). For a specification Γ , *cterm* is the smallest set where:

$$\frac{p \in \text{sterms } \Gamma \quad (\Gamma \text{ } pn)}{p \in \text{cterm } \Gamma} \quad \frac{pp \in \text{cterm } \Gamma \quad p \in \text{dterm } \Gamma \text{ } pp}{p \in \text{cterm } \Gamma}$$

It is also useful to define a local version independent of any specification.

Definition 8 (cterm_{sl}). Let *cterm_{sl}* be the smallest set defined by:

$$\begin{aligned} \text{cterm}_{sl} (p_1 \oplus p_2) &= \text{cterm}_{sl} p_1 \cup \text{cterm}_{sl} p_2, \\ \text{cterm}_{sl} (\text{call}(pn)) &= \{\text{call}(pn)\}, \\ \text{cterm}_{sl} (\{\!\!\{ \} \!\!\} [u] p) &= \{\{\!\!\{ \} \!\!\} [u] p\} \cup \text{cterm}_{sl} p, \text{ and so on.} \end{aligned}$$

Including call terms ensures that $q \in \text{stermsl } p$ implies $q \in \text{cterm}_{sl} p$, which facilitates proofs. For wellformed Γ , *cterm_{sl}* allows an alternative definition of *cterm*,

$$\text{cterm } \Gamma = \{p \mid \exists pn. p \in \text{cterm}_{sl} (\Gamma \text{ } pn) \wedge \text{not-call } p\}. \quad (1)$$

While the original definition is convenient for developing the meta-theory, due to the accompanying induction principle, this one is more useful for systematically generating the set of control terms of a specification, and thus, we will see, sets of verification conditions. And, for wellformed Γ , we have as a corollary

$$\text{cterm } \Gamma = \{p \mid \exists pn. p \in \text{subterms } (\Gamma \text{ } pn) \wedge \text{not-call } p \wedge \text{not-choice } p\}, \quad (2)$$

where *subterms*, *not-call*, and *not-choice* are defined in the obvious way.

We show that *cterm* over-approximates the set of reachable control states.

Lemma 1. For wellformed Γ and automaton A where *control-within* Γ (*init* A) and *trans* $A = \text{seqp-sos } \Gamma$, if $(\xi, p) \in \text{reachable } A$ and $q \in \text{sterms } \Gamma \text{ } p$ then $q \in \text{cterm } \Gamma$.

The predicate *control-within* $\Gamma \text{ } \sigma = \forall (\xi, p) \in \sigma. \exists pn. p \in \text{subterms } (\Gamma \text{ } pn)$ serves to state that the initial control state is within the specification.

3.2 Basic proof rule and invariants

Using the definition of invariance (Definition 2), we can state a basic property of an instance of the toy process:

$$\text{ptoy } i \models \text{onl } \Gamma_{\text{Toy}} (\lambda(\xi, l). l \in \{\text{PToy-:2}.. \text{PToy-:8}\} \longrightarrow \text{nhip } \xi = \text{ip } \xi), \quad (3)$$

This invariant states that between the lines labelled *PToy-:2* and *PToy-:8*, that is, after the assignment of *PToy-:1* until before the assignment of *PToy-:8*, the values of *nhip* and *ip* are equal; *onl* $\Gamma \text{ } P$, defined as $\lambda(\xi, p). \forall l \in \text{labels } \Gamma \text{ } p. P(\xi, l)$, extracts labels from control states.⁶ Invariants like these are solved using a procedure whose soundness is justified as a theorem. The proof exploits (1) and Lemma 1.

⁶ Using labels in this way is standard, see, for instance, [13, Chap. 1], or the ‘assertion networks’ of [18, §2.5.1]. Isabelle rapidly dispatches all the uninteresting cases.

Theorem 1. *To prove $A \models (I \rightarrow) \text{ onl } \Gamma P$, where wellformed Γ , simple-labels Γ , control-within Γ (init A), and $\text{trans } A = \text{seqp-sos } \Gamma$, it suffices*

- (init) *for arbitrary $(\xi, p) \in \text{init } A$ and $l \in \text{labels } \Gamma p$, to show $P(\xi, l)$, and,*
- (step) *for arbitrary $p \in \text{ctermst } (\Gamma \text{ pn})$, but not-call p , and $l \in \text{labels } \Gamma p$, given that $p \in \text{sterms } \Gamma pp$ for some $(\xi, pp) \in \text{reachable } A l$, to assume $P(\xi, l)$ and $l a$, and then for any (ξ', q) such that $((\xi, p), a, (\xi', q)) \in \text{seqp-sos } \Gamma$ and $l' \in \text{labels } \Gamma q$, to show $P(\xi', l')$.*

Here, simple-labels $\Gamma = \forall \text{pn}. \forall p \in \text{subterms } (\Gamma \text{ pn}). \exists ! l. \text{labels } \Gamma p = \{l\}$: each control term must have exactly one label, that is, \oplus terms must be labelled consistently.

We incorporate this theorem into a tactic that (1) applies the introduction rule, (2) replaces $p \in \text{ctermst } (\Gamma \text{ pn})$ by a disjunction over the values of pn , (3) applies Definition 8 and repeated simplifications of Γ 's and eliminations on disjunctions to generate one subgoal (verification condition) for each control term, (4) replaces control term derivatives, the subterms in Definition 6, by fresh variables, and, finally, (5) tries to solve each subgoal by simplification. Step 4 replaces potentially large control terms by their (labelled) heads, which is important for readability and prover performance. The tactic takes as arguments a list of existing invariants to include after having applied the introduction rule and a list of lemmas for trying to solve any subgoals that survive the final simplification. There are no schematic variables in the subgoals and we benefit greatly from Isabelle's `PARALLEL_GOALS` tactical [22].

In practice, one states an invariant, applies the tactic, and examines the resulting goals. One may need new lemmas for functions over the data state or explicit proofs for difficult goals. That said, the tactic generally dispatches the uninteresting goals, and the remaining ones typically correspond with the cases treated explicitly in manual proofs [4].

For step invariants, we show a counterpart to Theorem 1, and declare it to the tactic. Then we can show, for our example, that the value of `no` never decreases:

$$\text{ptoy } i \models (\lambda((\xi, -), -, (\xi', -)). \text{no } \xi \leq \text{no } \xi').$$

4 Open invariance

The analysis of network protocols often requires ‘inter-node’ invariants, like

$$\begin{aligned} \text{wf-net-tree } n \implies \text{closed } (\lambda i. \text{ptoy } i \ll \text{qmsg } n) \models \\ \text{netglobal } (\lambda \sigma. \forall i. \text{no } (\sigma i) \leq \text{no } (\sigma (\text{nhp } (\sigma i))))), \quad (4) \end{aligned}$$

which states that, for any `net-tree` with disjoint node addresses (`wf-net-tree n`), the value of `no` at a node is never greater than its value at the ‘next hop’—the address in `nhp`. This is a property of a global state σ mapping addresses to corresponding data states. Such a global state is readily constructed with:

$$\begin{aligned} \text{netglobal } P &= \lambda s. P (\text{default toy-init } (\text{netlift fst } s)), \\ \text{default df } f &= (\lambda i. \text{case } f \text{ of None } \Rightarrow \text{df } i \mid \text{Some } s \Rightarrow s), \text{ and} \\ \text{netlift sr } (s_R^i) &= [i \mapsto \text{fst } (\text{sr } s)] \\ \text{netlift sr } (s_{\parallel t}) &= \text{netlift sr } s \text{ ++ netlift sr } t. \end{aligned}$$

The applications of `fst` elide the state of `qmsg` and the protocol's control state.⁷

While we can readily state inter-node invariants of a complete model, showing them compositionally is another issue. Sections 4.1 and 4.2 present a way to state and prove such invariants at the level of sequential processes—that is, with only `ptoy i` left of the turnstile. Sections 4.3 and 4.4 present, respectively, rules for lifting such results to network models and for recovering invariants like (4).

4.1 The open model

Rather than instantiate the '`s`' of ('`s`', '`p`', '`l`') `seqp` with elements ξ of type `state`, our solution introduces a global state σ of type `ip` \Rightarrow `state`. This necessitates a stack of new SOS rules that we call the *open model*; Figure 5 shows some representatives.

The rules of `oseqp-sos` are parameterized by an address `i` and constrain only that entry of the global state, either to say how it changes ($\sigma' i = u (\sigma i)$) or that it does not ($\sigma' i = \sigma i$). The rules for `oparp-sos` only allow the first sub-process to constrain σ . This choice is disputable: it precludes comparing the states of `qmsgs` (and any other local filters) across a network, but it also simplifies the mechanics and use of this layer of the framework.⁸ The sets `onode-sos` and `opnet-sos` need not be parameterized since they are generated inductively from lower layers. Together they constrain subsets of elements of σ . This occurs naturally for rules like those for `arrive` and `*cast`, where the synchronous communication serves as a conjunction of constraints on sub-ranges of σ . But for others that normally only constrain a single element, like those for τ , assumptions ($\forall j \neq i. \sigma' j = \sigma j$) are introduced here and later dispatched (Section 4.4). The rules for `ocnet-sos`, not shown, are similar—elements not addressed within a model may not change.

The stack of operators and model layers described in Section 2 is refashioned to use the new transition rules and to distinguish the global state, which is preserved as the `fst` element across layers, from the local state elements which are combined in the `snd` element as before.

For instance, a sequential instance of the toy protocol is defined as

$$\text{optoy } i = \langle \text{init} = \{(\text{toy-init}, \Gamma_{\text{Toy}} \text{PToy})\}, \text{trans} = \text{oseqp-sos } \Gamma_{\text{Toy}} i \rangle,$$

combined with the standard `qmsg` process using the operator

$$\begin{aligned} s \langle i \text{ t} = \langle \text{init} = \{(\sigma, (s_l, t_l)) \mid (\sigma, s_l) \in \text{init } s \wedge t_l \in \text{init } t\}, \\ \text{trans} = \text{oparp-sos } i (\text{trans } s) (\text{trans } t) \rangle \rangle, \end{aligned}$$

and lifted to the node level via the open node constructor

$$\langle i : \text{onp} : R_i \rangle_o = \langle \text{init} = \{(\sigma, s_{R_i}^i) \mid (\sigma, s) \in \text{init } \text{onp}\}, \text{trans} = \text{onode-sos } (\text{trans } \text{onp}) \rangle.$$

Similarly, to map a `net-tree` term to an open model we define:

$$\begin{aligned} \text{opnet onp } \langle i : R_i \rangle &= \langle i : \text{onp } i : R_i \rangle_o \\ \text{opnet onp } (p_1 \parallel p_2) &= \langle \text{init} = \{(\sigma, s_1 \parallel s_2) \mid (\sigma, s_1) \in \text{init } (\text{opnet onp } p_1) \\ &\quad \wedge (\sigma, s_2) \in \text{init } (\text{opnet onp } p_2) \\ &\quad \wedge \text{net-ips } s_1 \cap \text{net-ips } s_2 = \emptyset\}, \\ &\quad \text{trans} = \text{opnet-sos } (\text{trans } (\text{opnet onp } p_1)) (\text{trans } (\text{opnet onp } p_2)) \rangle. \end{aligned}$$

⁷ The formulation here is a technical detail: `sr` corresponds to `netlift` as `np` does to `pnet`.

⁸ The treatment of the other layers is completely independent of this choice.

$$\begin{array}{c}
\frac{\sigma' i = u(\sigma i)}{((\sigma, \{\} \llbracket u \rrbracket p), \tau, (\sigma', p)) \in \text{oseqp-sos } \Gamma \ i} \quad \frac{((\sigma, p), a, (\sigma', p')) \in \text{oseqp-sos } \Gamma \ i}{((\sigma, p \oplus q), a, (\sigma', p')) \in \text{oseqp-sos } \Gamma \ i} \\
\\
\frac{\sigma' i = \sigma i}{((\sigma, \{\} \text{unicast}(s_{ip}, s_{msg}) \cdot p \triangleright q), \text{unicast}(s_{ip}(\sigma i)) (s_{msg}(\sigma i)), (\sigma', p)) \in \text{oseqp-sos } \Gamma \ i} \\
\text{(a) Sequential processes: examples from oseqp-sos.} \\
\\
\frac{((\sigma, s), \text{receive } m, (\sigma', s')) \in S \quad (t, \text{send } m, t') \in T}{((\sigma, (s, t)), \tau, (\sigma', (s', t')) \in \text{oparp-sos } i \ S \ T} \\
\text{(b) Parallel processes: example from oparp-sos.} \\
\\
\frac{((\sigma, s), \text{receive } m, (\sigma', s')) \in S}{((\sigma, s_R^i), \{i\} \neg \emptyset : \text{arrive}(m), (\sigma', s_R^{i'}) \in \text{onode-sos } S} \quad \frac{((\sigma, s), \tau, (\sigma', s')) \in S \quad \forall j \neq i. \sigma' j = \sigma j}{((\sigma, s_R^i), \tau, (\sigma', s_R^{i'})) \in \text{onode-sos } S} \\
\text{(c) Nodes: examples from onode-sos.} \\
\\
\frac{((\sigma, s), H \neg K : \text{arrive}(m), (\sigma', s')) \in S \quad ((\sigma, t), H' \neg K' : \text{arrive}(m), (\sigma', t')) \in T}{((\sigma, s \parallel t), (H \cup H') \neg (K \cup K') : \text{arrive}(m), (\sigma', s' \parallel t')) \in \text{opnet-sos } S \ T} \\
\text{(d) Partial networks: example from opnet-sos.}
\end{array}$$

Fig. 5: SOS rules for the open model (cf. Figures 1, 2, 3, and 4)

This definition is non-empty only for well-formed *net-trees* (*net-ips* gives the set of node addresses in the state of a partial network). Including such a constraint within the open model, rather than as a separate assumption like the *wf-net-tree* n in (4), eliminates an annoying technicality from the inductions described in Section 4.3. As with the extra premises in the open SOS rules, we can freely adjust the open model to facilitate proofs but each ‘encoded assumption’ becomes an obligation to be discharged in the transfer lemma of Section 4.4.

An operator for adding the last layer is also readily defined by

$$\text{oclosed } A = A(\text{trans} := \text{ocnet-sos}(\text{trans } A)),$$

giving all the definitions necessary to turn a standard model into an open one.

4.2 Open invariants

The basic definitions of reachability and invariance, Definitions 1–3, apply to open models, but constructing a compositional proof requires considering the effects of both synchronized and interleaved actions of possible environments.

Definition 9 (open reachability). *Given an automaton A and assumptions S and U over, respectively, synchronized and interleaved actions, $\text{oreachable } A \ S \ U$ is the smallest set defined by the rules:*

$$\begin{array}{c}
\frac{(\sigma, p) \in \text{init } A}{(\sigma, p) \in \text{oreachable } A \ S \ U} \quad \frac{(\sigma, p) \in \text{oreachable } A \ S \ U \quad U \ \sigma \ \sigma'}{(\sigma', p) \in \text{oreachable } A \ S \ U} \\
\\
\frac{(\sigma, p) \in \text{oreachable } A \ S \ U \quad ((\sigma, p), a, (\sigma', p')) \in \text{trans } A \quad S \ \sigma \ \sigma' \ a}{(\sigma', p') \in \text{oreachable } A \ S \ U}
\end{array}$$

In practice, we use restricted forms of the assumptions S and U , respectively,

$$\text{otherwith } E \mid \sigma \mid \sigma' \mid a = (\forall i. i \notin N \longrightarrow E(\sigma \mid i)(\sigma' \mid i)) \wedge \mid \sigma \mid a, \quad (5)$$

$$\text{other } F \mid N \mid \sigma \mid \sigma' = \forall i. \text{ if } i \in N \text{ then } \sigma' \mid i = \sigma \mid i \text{ else } F(\sigma \mid i)(\sigma' \mid i). \quad (6)$$

The former permits the restriction of possible environments (E) and also the extraction of information from shared actions (\mid). The latter restricts (F) the effects of interleaved actions, which may only change non-local state elements.

Definition 10 (open invariance). *Given an automaton A and assumptions S and U over, respectively, synchronized and interleaved actions, a predicate P is an open invariant, denoted $A \models (S, U \rightarrow) P$, iff $\forall s \in \text{oreachable } A \mid S \mid U. P \mid s$.*

It follows easily that existing invariants can be made open: most invariants can be shown in the basic context but still exploited in the more complicated one.

Lemma 2. *Given an invariant $A \models (I \rightarrow) P$ where $\text{trans } A = \text{seqp-sos } \Gamma$, and any F , there is an open invariant $A' \models (\lambda -. I, \text{other } F \{i\} \rightarrow) (\lambda(\sigma, p). P(\sigma \mid i, p))$ where $\text{trans } A' = \text{oseqp-sos } \Gamma \mid i$, provided that $\text{init } A = \{(\sigma \mid i, p) \mid (\sigma, p) \in \text{init } A'\}$.*

Open step invariance and a similar transfer lemma are defined similarly. The meta theory for basic invariants is also readily adapted, in particular,

Theorem 2. *To show $A \models (S, U \rightarrow) \text{onl } \Gamma \mid P$, in addition to the conditions and the obligations (**init**) and (**step**) of Theorem 1, suitably adjusted, it suffices,*

(env) *for arbitrary $(\sigma, p) \in \text{oreachable } A \mid S \mid U$ and $l \in \text{labels } \Gamma \mid p$, to assume both $P(\sigma, l)$ and $U \mid \sigma \mid \sigma'$, and then to show $P(\sigma', l)$.*

This theorem is declared to the tactic described in Section 3.2 and proofs proceed as before, but with the new obligation to show invariance over interleaved steps.

We finally have sufficient machinery to state (and prove) Invariant (4) at the level of a sequential process:

$$\text{optoy } i \models (\text{otherwith } \text{nos-inc } \{i\} (\text{orecvmsg msg-ok}), \text{other } \text{nos-inc } \{i\} \rightarrow) \quad (7) \\ (\lambda(\sigma, -). \text{no } (\sigma \mid i) \leq \text{no } (\sigma (\text{nhip } (\sigma \mid i)))) ,$$

where $\text{nos-inc } \xi \mid \xi' = \text{no } \xi \leq \text{no } \xi'$, orecvmsg applies its given predicate to receive actions and is otherwise true, $\text{msg-ok } \sigma (\text{pkt } (\text{data}, \text{src})) = (\text{data} \leq \text{no } (\sigma \mid \text{src}))$, and $\text{msg-ok } \sigma (\text{newpkt } (\text{data}, \text{dst})) = \text{True}$. So, given that the variables no in the environment never decrease and that incoming pkts reflect the state of the sender, there is a relation between the local node and the next hop. Similar invariants occur in proofs of realistic protocols [4].

4.3 Lifting open invariants

The next step is to lift Invariant (7) over each composition operator of the open model. We mostly present the lemmas over oreachable , rather than those for open invariants and step invariants, which follow more or less directly.

The first lifting rule treats composition with the `qmsg` process. It mixes `oreachable` and `reachable` predicates: the former for the automaton being lifted, the latter for properties of `qmsg`. The properties of `qmsg`—only received messages are added to the queue and sent messages come from the queue—are shown using the techniques of Section 3.

Lemma 3 (qmsg lifting). *Given $(\sigma, (s, (q, t))) \in \text{oreachable } (A \langle \langle i \rangle \text{ qmsg} \rangle S \ U)$, where predicates $S = \text{otherwith } E \{i\} \text{ (orecvmsg } R)$ and $U = \text{other } F \{i\}$, and provided (1) $A \models (S, U \rightarrow) (\lambda((\sigma, -), -, (\sigma', -)). F(\sigma \ i) (\sigma' \ i))$, (2) for all $\xi, \xi', E \xi \xi'$ implies $F \xi \xi'$, (3) for all $\sigma, \sigma', m, \forall j. F(\sigma \ j) (\sigma' \ j)$ and $R \sigma \ m$ imply $R \sigma' \ m$, and, (4) F is reflexive, then $(\sigma, s) \in \text{oreachable } A \ S \ U$ and $(q, t) \in \text{reachable qmsg (recvmsg } (R \ \sigma))$, and furthermore $\forall m \in \text{set } q. R \ \sigma \ m$.*

The key intuition is that every message m received, queued, and sent by `qmsg` satisfies $R \ \sigma \ m$. The proof is by induction over `oreachable`. The R 's are preserved when the external environment acts independently (3, 4), when it acts synchronously (2), and when the local process acts (1, 3).

The rule for lifting to the node level adapts assumptions on receive actions (`orecvmsg`) to arrive actions (`oarrivmsg`).

Lemma 4 (onode lifting). *If, for all ξ and $\xi', E \xi \xi'$ implies $F \xi \xi'$, then given $(\sigma, s_R^i) \in \text{oreachable } (\langle i : A : R_i \rangle_o) \text{ (otherwith } E \{i\} \text{ (oarrivmsg } I)) \text{ (other } F \{i\})$ it follows that $(\sigma, s) \in \text{oreachable } A \text{ (otherwith } E \{i\} \text{ (orecvmsg } I)) \text{ (other } F \{i\})$.*

The sole condition is needed because certain node-level actions—namely `connect`, `disconnect`, and `$\emptyset \rightarrow \{i\} : \text{arrive}(m)$` —synchronize with the environment (giving $E \xi \xi'$) but appear to ‘stutter’ (requiring $F \xi \xi'$) relative to the underlying process.

The lifting rule for partial networks is the most demanding. The function `net-tree-ips`, giving the set of addresses in a `net-tree`, plays a key role.

Lemma 5 (opnet lifting). *Given $(\sigma, s \parallel t) \in \text{oreachable } (\text{opnet onp } (p_1 \parallel p_2)) \ S \ U$, where $S = \text{otherwith } E \text{ (net-tree-ips } (p_1 \parallel p_2)) \text{ (oarrivmsg } I)$, $U = \text{other } F \text{ (net-tree-ips } (p_1 \parallel p_2))$, and E and F are reflexive, for arbitrary $p \ i$ of the form $\langle i : \text{onp } i : R \rangle_o$, $p \ i \models (\lambda \sigma \ -. \text{oarrivmsg } I \ \sigma, \text{other } F \{i\} \rightarrow) (\lambda((\sigma, -), a, (\sigma', -)). \text{castmsg } (I \ \sigma) \ a)$, and similar step invariants for $E(\sigma \ i) (\sigma' \ i)$ and $F(\sigma \ i) (\sigma' \ i)$, then it follows that both $(\sigma, s) \in \text{oreachable } (\text{opnet onp } p_1) \ S_1 \ U_1$ and $(\sigma, t) \in \text{oreachable } (\text{opnet onp } p_2) \ S_2 \ U_2$, where S_1 and U_1 are over p_1 , and S_2 and U_2 are over p_2 .*

The proof is by induction over `oreachable`. The initial and interleaved cases are trivial. For the local case, given open reachability of (σ, s) and (σ, t) for p_1 and p_2 , respectively, and $((\sigma, s \parallel t), a, (\sigma', s' \parallel t')) \in \text{trans } (\text{opnet onp } (p_1 \parallel p_2))$, we must show open reachability of (σ', s') and (σ', t') . The proof proceeds by cases of a . The key step is to have stated the lemma without introducing cyclic dependencies between (synchronizing) assumptions and (step invariant) guarantees. For a synchronizing action like `arrive`, Definition 9 requires satisfaction of S_1 to advance in p_1 and of S_2 to advance in p_2 , but the assumption S only holds for addresses $j \notin \text{net-tree-ips } (p_1 \parallel p_2)$. This is why the step invariants required of nodes only assume `oarrivmsg` $I \ \sigma$ of the environment, rather than an S over node address $\{i\}$.

This is not unduly restrictive since the step invariants provide guarantees for individual local state elements and not between network nodes. The assumption $\text{oarrivmsg } l \sigma$ is never cyclic: it is either assumed of the environment for paired arrives, or trivially satisfied for the side that $\ast\text{casts}$. The step invariants are lifted from nodes to partial networks by induction over net-trees . For non-synchronizing actions, we exploit the extra guarantees built into the open SOS rules.

The rule for closed networks is similar to the others. Its important function is to eliminate the synchronizing assumption (S in the lemmas above), since messages no longer arrive from the environment. The conclusion of this rule has the form required by the transfer lemma of the next section.

4.4 Transferring open invariants

The rules in the last section extend invariants over sequential processes, like that of (7), to arbitrary, open network models. All that remains is to transfer the extended invariants to the standard model. We do so using a locale [12] $\text{openproc } np \text{ onp } sr$ where np has type $\text{ip} \Rightarrow ('s, 'm \text{ seq-action})$ automaton, onp has type $\text{ip} \Rightarrow ((\text{ip} \Rightarrow 'g) \times 'l, 'm \text{ seq-action})$ automaton, and sr has type $'s \Rightarrow 'g \times 'l$. The automata use the actions of Section 2.1 with arbitrary messages ($'m \text{ seq-action}$).

The openproc locale relates an automaton np to a corresponding ‘open’ automaton onp , where sr splits the states of the former into global and local components. Besides two technical conditions on initial states, this relation requires assuming $\sigma \cdot i = \text{fst}(sr \cdot s)$, $\sigma' \cdot i = \text{fst}(sr \cdot s')$ and $(s, a, s') \in \text{trans}(np \cdot i)$, and then showing $((\sigma, \text{snd}(sr \cdot s)), a, (\sigma', \text{snd}(sr \cdot s'))) \in \text{trans}(onp \cdot i)$ —that is, that onp simulates np . For our running example, we show $\text{openproc } ptoy \text{ optoy } id$, and then lift it to the composition with $qmsg$, using a generic relation on openproc locales.

Lemma 6 (transfer). *Given np , onp , and sr such that $\text{openproc } np \text{ onp } sr$, then for any wf-net-tree n and $s \in \text{reachable}(\text{closed}(pnet \text{ } np \text{ } n))$ ($\lambda\text{-True}$), it follows that*

$$(\text{default}(\text{someinit } np \text{ } sr) (\text{netlift } sr \text{ } s), \text{netlift } sr \text{ } s) \in \text{oreachable}(\text{oclosed}(opnet \text{ } onp \text{ } n)) (\lambda\text{-True}) U.$$

This lemma uses two openproc constants: $\text{someinit } np \text{ } sr \cdot i$ chooses an arbitrary initial state from np ($\text{SOME } x. x \in (\text{fst} \circ sr) \cdot \text{init}(np \cdot i)$), and

$$\begin{aligned} \text{netlift } sr \text{ } (s_R^i) &= (\text{snd}(sr \cdot s))_R^i \\ \text{netlift } sr \text{ } (s \parallel t) &= (\text{netlift } sr \text{ } s) \parallel (\text{netlift } sr \text{ } t). \end{aligned}$$

The proof of the lemma ‘discharges’ the assumptions incorporated into the open SOS rules. An implication from an open invariant on an open model to an invariant on the corresponding standard model follows as a corollary.

Summary. The technicalities of the lemmas in this and the preceding section are essential for the underlying proofs to succeed. The key idea is that through an open version of AWN where automaton states are segregated into global and local components, one can reason locally about global properties, but still, using the so called transfer and lifting results, obtain a result over the original model.

5 Concluding remarks

We present a mechanization of a modelling language for MANET and WMN protocols, including a streamlined adaptation of standard theory for showing invariants of individual reactive processes, and a novel and compositional framework for lifting such results to network models. The framework allows the statement and proof of inter-node properties. We think that many elements of our approach would apply to similarly structured models in other formalisms.

It is reasonable to ask whether the basic model presented in Section 2 could not simply be abandoned in favour of the open model of Section 4.1. But we believe that the basic model is the most natural way of describing what AWN means, proving semantic properties of the language, showing ‘node-only’ invariants, and, potentially, for showing refinement relations. Having such a reference model allows us to freely incorporate assumptions into the open SOS rules, knowing that their soundness must later be justified.

The Ad hoc On-demand Distance Vector (AODV) case study. The framework we present in this paper was successfully applied in the mechanization of a proof of loop freedom [6, §7] of the AODV protocol [17], a widely-used routing protocol designed for MANETs, and one of the four protocols currently standardized by the IETF MANET working group. The model has about 100 control locations across 6 different processes, and uses about 40 functions to manipulate the data state. The main property (loop freedom) roughly states that ‘a data packet is never sent round in circles without being delivered’. To establish this property, we proved around 400 lemmas. Due to the complexity of the protocol logic and the length of the proof, we present the details elsewhere [4]. The case study shows that the presented framework can be applied to verification tasks of industrial relevance.

Acknowledgments. We thank G. Klein and M. Pouzet for support and complaisance, and M. Daum for participation in discussions. Isabelle/jEdit [21], Sledgehammer [2], parallel processing [22], and the TPTP project [19] were invaluable.

NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

References

1. J. Bengtson and J. Parrow. Psi-calculi in Isabelle. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *TPHOLs’09*, volume 5674 of *LNCS*, pages 99–114. Springer, 2009.
2. J. C. Blanchette, S. Böhme, and L. C. Paulson. Extending Sledgehammer with SMT solvers. In N. Bjørner and V. Sofronie-Stokkermans, editors, *CADE-23*, volume 6803 of *LNCS*, pages 116–130. Springer, 2011.
3. T. Bourke. Mechanization of the Algebra for Wireless Networks (AWN). *Archive of Formal Proofs*, 2014. <http://afp.sf.net/entries/AWN.shtml>.
4. T. Bourke, R. J. van Glabbeek, and P. Höfner. A mechanized proof of loop freedom of the (untimed) AODV routing protocol, 2014. See authors’ webpages.

5. K. Chaudhuri, D. Doligez, L. Lamport, and S. Merz. Verifying safety properties with the TLA^+ proof system. In J. Giesl and R. Hähnle, editors, *IJCAR'10*, volume 6173 of *LNCS*, pages 142–148. Springer, 2010.
6. A. Fehnker, R. J. van Glabbeek, P. Höfner, A. McIver, M. Portmann, and W. L. Tan. A process algebra for wireless mesh networks used for modelling, verifying and analysing AODV. Technical Report 5513, NICTA, 2013. <http://arxiv.org/abs/1312.7645>.
7. A. Feliachi, M.-C. Gaudel, and B. Wolff. Isabelle/Circus: A process specification and verification environment. In R. Joshi, P. Müller, and A. Podelski, editors, *VSTTE'12*, volume 7152 of *LNCS*, pages 243–260. Springer, 2012.
8. W. Fokkink, J. F. Groote, and M. Reniers. Process algebra needs proof methodology. In *EATCS Bulletin 82*, pages 109–125, 2004.
9. T. Göthel and S. Glesner. An approach for machine-assisted verification of Timed CSP specifications. *Innovations in Systems and Software Engineering*, 6(3):181–193, 2010.
10. B. Heyd and P. Crégut. A modular coding of UNITY in COQ. In G. Goos, J. Hartmanis, J. Leeuwen, J. Wright, J. Grundy, and J. Harrison, editors, *TPHOLs'96*, volume 1125 of *LNCS*, pages 251–266. Springer, 1996.
11. D. Hirschhoff. A full formalisation of π -calculus theory in the Calculus of Constructions. In K. Schneider and J. Brandt, editors, *TPHOLs'07*, volume 4732 of *LNCS*, pages 153–169. Springer, 2007.
12. F. Kammüller, M. Wenzel, and L. C. Paulson. Locales: A sectioning concept for Isabelle. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Théry, editors, *TPHOLs'99*, volume 1690 of *LNCS*, pages 149–165. Springer, 1999.
13. Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer, 1995.
14. O. Müller. *A Verification Environment for I/O Automata Based on Formalized Meta-Theory*. PhD thesis, TU München, 1998.
15. T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
16. L. C. Paulson. The inductive approach to verifying cryptographic protocols. *J. Computer Security*, 6(1–2):85–128, 1998.
17. C. E. Perkins, E. M. Belding-Royer, and S. R. Das. Ad hoc on-demand distance vector (AODV) routing. RFC 3561 (Experimental), Network Working Group, 2003.
18. W.-P. de Roever, F. de Boer, U. Hannemann, J. Hooman, Y. Lakhnech, M. Poel, and J. Zwiers. *Concurrency Verification: Introduction to Compositional and Non-compositional Methods*. Cambridge Tracts in Theor. Comp. Sci. 54. CUP, 2001.
19. G. Sutcliffe. The TPTP problem library and associated infrastructure: The FOF and CNF parts, v3.5.0. *J. Automated Reasoning*, 43(4):337–362, 2009.
20. H. Tej and B. Wolff. A corrected failure divergence model for CSP in Isabelle/HOL. In J. S. Fitzgerald, C. B. Jones, and P. Lucas, editors, *FME'97*, volume 1313 of *LNCS*, pages 318–337. Springer, 1997.
21. M. Wenzel. Isabelle/jEdit—a prover IDE within the PIDE framework. In J. Jeuring, J. A. Campbell, J. Carette, G. Dos Reis, P. Sojka, M. Wenzel, and V. Sorge, editors, *Intelligent Computer Mathematics*, volume 7362 of *LNCS*, pages 468–471. Springer, 2012.
22. M. Wenzel. Shared-memory multiprocessing for interactive theorem proving. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *ITP'13*, volume 7998 of *LNCS*, pages 418–434. Springer, 2013.